Improvasher: a real-time mashup system for live musical input

Matthew E. P. Davies
INESC TEC
Sound and Music Computing Group
Porto, Portugal
mdavies@inescporto.pt

Adam M. Stark
Independent Researcher
London, United Kingdom
adamstark.uk@gmail.com

Fabien Gouyon
INESC TEC
Sound and Music Computing Group
Porto, Portugal
fgouyon@inescporto.pt

Masataka Goto
National Institute of Advanced
Industrial Science and
Technology (AIST)
Tsukuba, Ibaraki, Japan
m.goto@aist.go.jp

ABSTRACT

In this paper we present *Improvasher* a real-time musical accompaniment system which creates an automatic mashup to accompany live musical input. Improvasher is built around two music processing modules, the first, a performance following technique, makes beat-synchronous predictions of chroma features from a live musical input. The second, a music mashup system, determines the compatibility between beat-synchronous chromagrams from different pieces of music. Through the combination of these two techniques, a real-time predictive mashup can be generated towards a new form of automatic accompaniment for interactive musical performance.

Keywords

Automatic accompaniment, music mashups, real-time musical interaction, Max.

1. INTRODUCTION

One of the earliest research areas in music information retrieval (MIR) addressed the topic of automatic musical accompaniment [4] to enable computers to play along with musicians in a live performance context. While much research in this area is built around *a priori* knowledge of the input piece through access to the musical score [3], a recent system [12] addressed the automatic accompaniment problem for improvised music where no score exists. In this so called "performance following" system, the output of a real-time beat-synchronous chromagram was passed to a dynamic programming search method to make a prediction of future chroma content based on analysis of repeated structure from earlier in the live performance.

A related research topic which address the combination of music signals, but in a different context, is music mashups. The goal of computational systems which attempt to automate the music mashup creation process is to determine the best match between a target song and some number of possible candidate songs, such that two (or more) musical signals can be seamlessly mixed together to create an interesting listening experience [11]. To this end, a recent mashup system [5] used time-stretching and pitch-shifting to explore a large search space of possible matches whereby songs were not only beat-matched to enable temporal synchronisation, but were also modulated in key to create harmonically compatible mixes for songs originally in different keys. Through the use of phrase-level structural segmentation on the input song, a multi-song mashup could be created by finding the best match per phrase from a set of candidate songs, and piecing the mashup together section by section.

In this work we seek to extend the creative musical possibilities available from the predictive aspect of the performance following system, by attempting to fuse automatic accompaniment with music mashups. Our goal is to create a new kind of musical interaction which we call "automashup accompaniment". Given a live musical input (e.g. from a musician playing the guitar), we use the performance follower to predict the chroma (i.e. harmonic) content for the next beat in the live input signal, and send this chroma information to the mashup system to determine the best matching beat slice from a set of candidate songs. Once the beat slice with the highest compatibility has been found, the corresponding audio content is played back in real-time to accompany the live input. Proceeding in this way, beat-by-beat, our system *Improvasher*, can create a real-time mashup accompaniment for the live musician. An overview of our real-time mashup concept is shown in Figure 1.

Within the field of computer music, our goal of automashup accompaniment can be considered similar to concatenative synthesis [10, 13] since we attempt to use some properties of a input signal to drive the some other audio content – in this case a matching of beat-synchronous harmonic content. The main distinction here is that we work at a larger time-scale (i.e. at the beat level) rather than at the frame-level of audio signals to allow the musical content comprising the real-time mashup to be recognisable to the performer and to listeners. In this light, perhaps the most closely related system to ours is Jehan's cross-synthesis technique [7]. However, our approach is designed to operate in a predictive real-time context, rather than offline. Furthermore, we require the matching content to be played back to accompany the live input, as opposed to Jehan's goal of approximate reconstruction of one song from another.

Improvasher is implemented using Max for the perfor-

NIME'14, June 30 – July 03, 2014, Goldsmiths, University of London, UK.

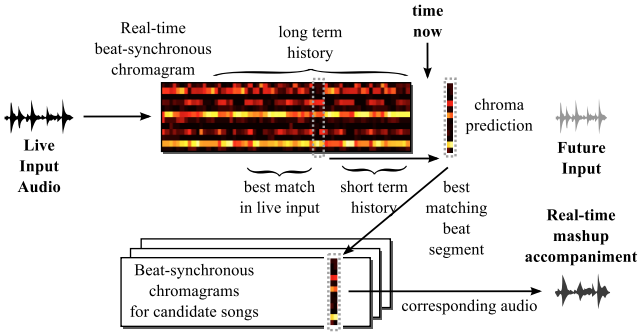

Figure 1: The concept of the real-time mashup system. Performance following analysis is used to predict the chroma content of the next beat of the live input. The best matching chroma to this prediction in a set of candidate songs is found, and played back to accompany the input.

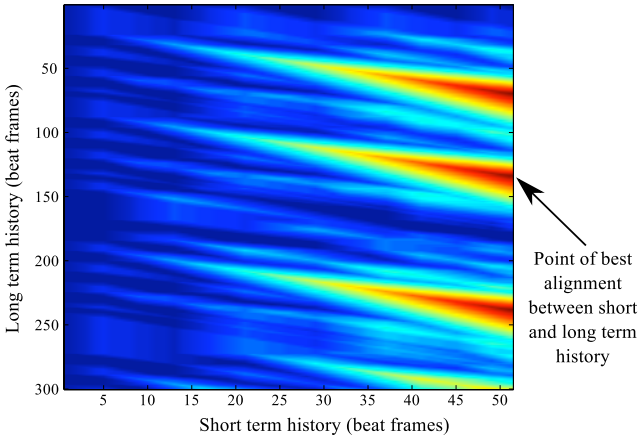

Figure 2: An example of the alignment matrix calculated using the long term (300 beat) and short term (52 beats) histories. There are several points of strong alignment, the strongest of which is indicated by the arrow.

mance following module, and a standalone C++ application for the mashup module, where the two are connected via Open Sound Control (OSC).

The remainder of this paper is structured as follows. In Section 2 we summarise the technical implementation of Improvasher and highlight two possible usage scenarios. In Section 3 we summarise the main contributions of the paper and propose areas for future work.

2. IMPROVASHER

In this section we outline the main components which comprise Improvasher, beginning with the performance following module, after which we discuss the mashup module, the overall system implementation and usage scenarios.

2.1 Performance following module

The live input processing module of the system is the "performance following" algorithm presented in [12]. As a front end, using a click track or beat tracker, this algorithm calculates a real-time sequence of beat-synchronous chromagram features by summing frame-level chroma estimations between beats. This means that at the ith beat b_i, we have the beat-synchronous chromagram C_k for the kth inter-beat interval $[b_{i-1}, b_i]$.

This information only tells us about what has *just happened*, and so in order to have some information on the harmonic content of the *current* inter-beat interval, the algorithm attempts to predict the next beat-synchronous chroma vector $\hat{C}_{k+1}$, based upon the sequence of chromagram vectors observed to date in the live musical input.

The prediction process involves comparing the long term history (e.g. the previous 150 beat-synchronous chroma vectors) to the short term history (e.g. the previous 16 beat-synchronous chroma vectors). The intention is to find an example of a repetition (or partial repetition) of the short term history at a previous point in the long term history. If such an example can be found, we can then inspect the beat-synchronous chromagram over the beats that follow on from this point and use them as predictions of future harmonic content.

In order to achieve this, an alignment matrix is calculated between the long term and short term histories, as shown in Figure 2. This is used to estimate the inter-beat interval in the past where the short term history (i.e. what is currently being played by the musician) best aligns with the long term history. Finally, the chroma vector of the beat-synchronous chromagram which immediately follows the point of best alignment is used as our prediction $\hat{C}_{k+1}$.

2.2 Mashup Module

The goal of the mashup module within Improvasher is to determine the compatibility or so-called "mashability" between harmonic content from the input and some set of candidate songs which could form part of the real-time mashup. The estimation of mashability is built around the measurement of cosine similarity between beat-synchronous chromagrams - in our case, between the live performance and pre-calculated beat-synchronous chromagrams for the set of candidate songs. In contrast to the mashup system in [5] which measured mashability over phrase-level sections (e.g. 16 or 32 beats) in the input signal, we perform the matching over a much shorter time-scale, i.e. the one beat at a time that the performance follower predicts on the fly. We currently restrict the range of the prediction to a single beat to ensure the highest reliability and consistency in the predicted chroma content used.

The input to the mashup module is the predicted chroma vector $\hat{C}_{k+1}$ given by the performance follower module at the onset of the ith beat b_i in the music. To find the best matching beat-slice from the set of N candidate songs, we measure the cosine similarity, S, between $\hat{C}_{k+1}$ and all beats, p, of the beat-synchronous chromagrams C^n across all candidate songs,

$$S_p^n = \frac{\hat{C}_{k+1} \cdot C_p^n}{||\hat{C}_{k+1}||\,||C_p^n||}. \quad (1)$$

Once this brute-force search has been completed across all beat frames p and candidate songs n, the song, $n_{\max}$, with the highest cosine similarity is found as follows

$$n_{\max} = \underset{n}{\text{argmax}}(\max(S_p^n)) \quad (2)$$

and then the corresponding beat slice $p_{\max}$ is found from $n_{\max}$ as

$$p_{\max} = \underset{p}{\text{argmax}}(S_p^{n_{\max}}). \quad (3)$$

Having found the best matching beat frame, we then immediately playback the corresponding audio slice so that it is beat-synchronised with the live input. At the onset of the next beat, and hence the next predicted beat chroma vector, the process repeats as before. To smooth out any potential

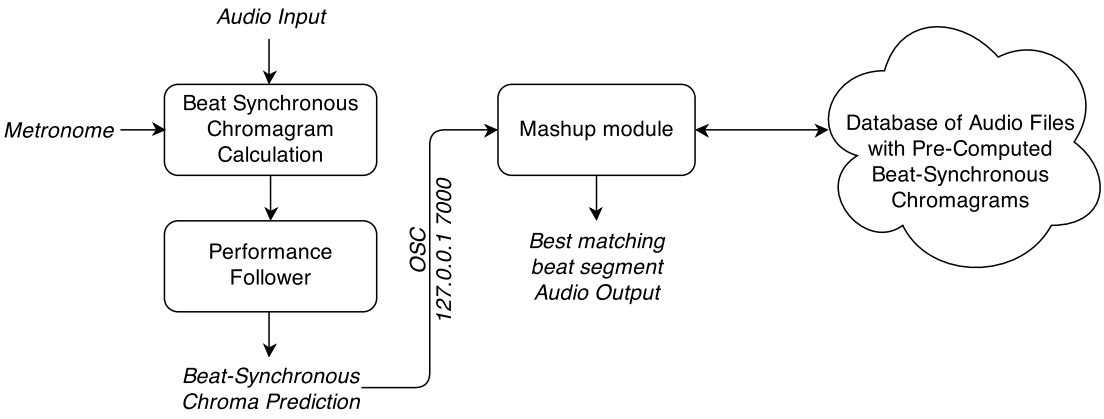

Figure 3: An overview of the Improvasher architecture showing the inputs, outputs and interconnections between components.

discontinuities which might arise from the concatenation of non-consecutive beat slices of audio, we implement a cross-fade over the first 10ms of each new beat.

2.3 Implementation

The Improvasher system comprises two main parts, the performance follower module and the mashup module. The core of the Improvasher system is an extension of the performance follower Max patch from [12]. This handles the incoming audio stream, the creation of the beat-synchronous chromagram and the prediction of future chroma vectors at the onset of each new beat. The mashup module is implemented as a separate standalone C++ application built using the Juce Library [9] which receives a chroma vector as input, determines mashability and then triggers playback of the audio corresponding to the best matching beat slice from the set of candidate songs. This transmission of chroma vectors from the performance follower to the mashup module happens via OSC (open sound control). An overview of the complete Improvasher system is shown in Figure 3. A corresponding screenshot of the Improvasher Max patch is shown in Figure 4.

For this prototype implementation of Improvasher we impose three constraints to simplify the processing and reduce the computational burden:

- For all candidate songs, we pre-compute the beat-synchronous chromagram representation so this can be pre-loaded into Improvasher prior to the start of the live musical performance, thus negating the need to analyse the set of candidate songs more than once.
- We force the tempo of the live input to be fixed by having the musician play along with a metronome set to 120 beats per minute (bpm). This removes the need to include real-time beat tracking which could introduce errors, and allows us to time-stretch all candidate songs to be at 120 bpm (i.e. exactly 500 ms per beat slice) before run-time. For time-stretching and pitch-shifting we use the open source Rubberband Library [2].
- To reflect the ability of the mashup system in [5] to find matches between songs in different keys, we pre-compute the pitch-shifted versions of all candidate songs, over a range of ±3 semitones. These key-shifted versions effectively appear as *different* candidate songs, but this prevents the need perform pitch-shifting at run time.

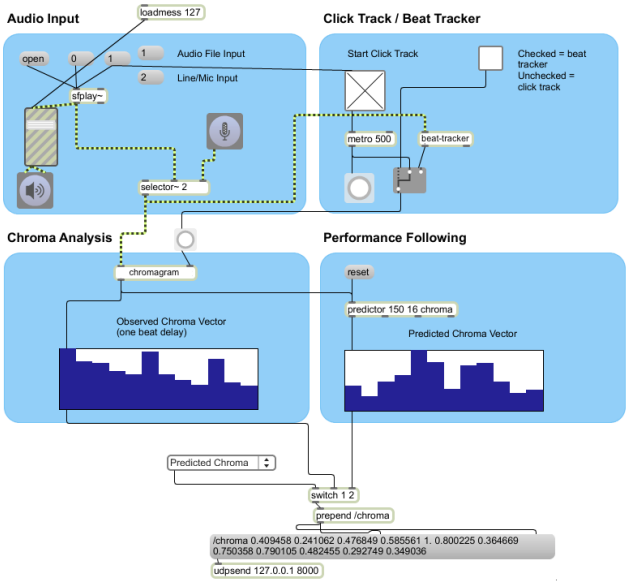

Figure 4: A screenshot of the Improvasher patch for Max.

By constraining Improvasher in this way, we need only retrieve the audio corresponding to the best matching beat slice and commence playback without any additional processing, thus minimising the computational cost while maintaining the majority of the core functionality of the mashup system in [5]. We revisit these constraints within future work in Section 3.

2.4 Usage

Based on our initial experiments we have explored two main usage scenarios for Improvasher. The first involves creating a real-time mashup to accompany a live music performance, e.g. from a guitar. As specified above, we simplify the creation of the real-time mashup accompaniment by restricting the performer to play along to a metronome, thus eliminating the need for real-time automatic beat tracking. To exert some control over how the mashup can sound, the musician can specify a subset of the pre-analysed candidate songs, for instance only using examples of solo bass playing, or only music from a particular style (e.g. house music). Alternatively, the musician can choose to include all candidate songs to create more experimental mashups.

The second usage scenario is a direct result of the pre-

computation of time-stretched versions of the candidate songs. Since we know the precise locations of the beats ahead of time, any of these candidate songs can themselves be the input to the system coupled to the 120 bpm metronome used for the live musical input. In this way, we can use Improvasher as kind a of real-time DJ mashup tool. Once again, the choice of remaining candidate songs can be customised, and the user can control when the mashup is generated by interacting with the "Start Click Track" button in Figure 4. To demonstrate the usage of Improvasher in both usage scenarios, we provide a website with video examples [6].

3. DISCUSSION AND CONCLUSIONS

In this paper we have presented Improvasher, a system for creating real-time music mashups as accompaniment to a live musical input. The main contribution of this work is in the application of a predictive model of harmonic content to drive a real-time mashup accompaniment for live musical input. Our initial experiences using Improvasher, particularly with a live guitar input have shown that we can use the musical performance to "guide" the selection of harmonically related beat segments by playing different chords. While the output of the system can sound very different to the input, it appears to exhibit some musical connection with the live input.

In the current implementation of Improvasher we imposed several constraints. In particular, we fixed the tempo of the input using a metronome and pre-computed time-stretched and pitch-shifted candidates for use in the the mashup accompaniment. To expand the flexibility for live musical input – to allow for more expressive musical timing, our main goal in future work will be to relax these constraints and directly incorporate a real-time beat tracking front-end. While this increases the complexity of the front-end analysis, and requires accurate estimation of beats in real-time, it also generates the need to undertake time-stretching (and potentially pitch-shifting) on the fly, which we consider a significant technical challenge.

Beyond the inclusion of time-varying analysis into Improvasher, we intend to explore several further extensions. We will examine the effect of extending the duration of the prediction of harmonic content made by the performance following module. This could allow for real-time bar-synchronous mashups, or the use of longer phrase-level structure, and thereby make the components used in the mashup more recognisable, potentially increasing the overall enjoyment of the musical result for listeners. Following the techniques presented in the earGram system [1] for concatenative music composition, we will also investigate the nature of the transitions between contiguous sections used in the mashup towards imposing more structure and internal coherence. In addition, we plan to incorporate more features for estimating mashability, in particular those which can capture timbral and rhythmic characteristics of the music to offer users more than just harmonic compatibility in mashup creation.

Finally, we will address the interface, or possible interfaces of Improvasher. Within the context of the real-time DJ mashup usage scenario, we plan to build a Max for Live device for direct integration into Abelton Live, thus removing the need for separate Max and Juce applications. Furthermore, we believe that tangible interfaces (e.g. [8]) could allow users to control and interact in different ways with the content used in the real-time mashup. We believe that these extensions will greatly enhance the creative possibilities of Improvasher and the field of real-time mashups in general.

Acknowledgments

This research was partially funded by the FCT post-doctoral grant (SFRH/BPD/88722/2012), the Media Arts and Technologies project (MAT), NORTE-07-0124-FEDER-000061, financed by the North Portugal Regional Operational Programme (ON.2 – O Novo Norte), under the National Strategic Reference Framework (NSRF), through the European Regional Development Fund (ERDF), and by national funds, through the Portuguese funding agency, Fundação para a Ciência e a Tecnologia (FCT), and by OngaCrest, CREST, JST.

4. REFERENCES

[1] G. Bernardes, C. Guedes, and B. Pennycook. Eargram: An application for interactive exploration of concatenative sound synthesis in Pure Data. In M. Aramaki, M. Barthet, R. Kronland-Martinet, and S. Ystad, editors, *From Sounds to Music and Emotions*, volume 7900 of *Lecture Notes in Computer Science*, pages 110–129. Springer Berlin Heidelberg, 2013.

[2] C. Cannam. Rubber Band Library. http://breakfastquay.com/rubberband/.

[3] A. Cont. A coupled duration-focused architecture for realtime music to score alignment. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(6):974–987, 2010.

[4] R. Dannenberg. An on-line algorithm for real-time accompaniment. In *Proceedings of the International Computer Music Conference*, pages 193–198, 1984.

[5] M. E. P. Davies, P. Hamel, K. Yoshii, and M. Goto. AutoMashUpper: An automatic multi-song mashup system. In *Proceedings of the 14th International Society for Music Information Retrieval Conference*, pages 575–580, 2013.

[6] M. E. P. Davies, A. M. Stark, F. Gouyon, and M. Goto. Improvasher web site. http://smc.inescporto.pt/technologies/improvasher/.

[7] T. Jehan. Event-synchronous music analysis/synthesis. In *Proceedings of the 7th International Conference on Digital Audio Effects (DAFx-04)*, pages 361–366, 2004.

[8] S. Jordà, G. Geiger, M. Alonso, and M. Kaltenbrunner. The reacTable: Exploring the synergy between live music performance and tabletop tangible interfaces. In *Proceedings of the 1st International Conference on Tangible and Embedded Interaction*, pages 139–146, 2007.

[9] Juce. Juce Cross-Platform C++ Library. http://www.juce.com.

[10] D. Schwarz, G. Beller, B. Verbugghe, and S. Britton. Real-time corpus-based concatenative synthesis with CataRT. In *Proceedings of the 9th International Conference on Digital Audio Effects (DAFx-06)*, pages 279–282, 2006.

[11] J. Shiga. Copy-and-Persist: The Logic of Mash-Up Culture. *Critical Studies in Media Communication*, 24(2):93–114, 2007.

[12] A. M. Stark and M. D. Plumbley. Performance following: Real-time prediction of musical sequences without a score. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(1):190–199, 2012.

[13] B. Sturm. Adaptive concatenative sound synthesis and its application to micromontage composition. *Computer Music Journal*, 30(4):44–66, 2006.